

E-Center Data Retrieval Service (DRS)

04/03/2011

Version 3.2

Previous Versions: 1.006a, 2.001a, 2.2b,2.3, 3.0,3.1

Maxim Grigoriev

Introduction

The E-center consists of collection of the loosely coupled applications. It will be kept this way in order to allow independent development, easy maintenance and support. In order to make these applications to communicate we assume REST based web services approach. How data is cached, retrieved from remote perfSONAR-PS(pS-PS) services, archived, pre-fetched behind the scenes is not covered in this document. In general it is valid to assume that all data is coming from the certain kind of the remote pS-PS service – Measurement Archive (MA).

Any resource representing the network measurement data will be easily addressed via simple URL and retrieved by sending HTTP request – for example to get pinger data for the end to end path with source and destination pair IP1=131.225.1.1 and IP2=130.224.3.3 one may send this query:

GET http://<ecenter server hostname>/data/pinger.json?src_ip=131.225.1.1&dst_ip=130.224.3.3&start=2010-05-02%20:01:02&end=2010-05-02%20:01:02

In addition, this document provides description of the more convenient data retrieval request targeted on the complete End-to-End path coverage.

Basic Concepts

The API uses HTTP GET request type and will return response encoded as JSON, YAML or XML data and HTTP status codes as:

- **200 OK:** Success, JSON encoded response is expected
- **400 Bad Request:** The request was invalid. An accompanying error message will explain why, see below for more details.
- **404 Not Found:** The URL requested is invalid or the resource requested, such as a metadata id, does not exist.
- **500 Internal Server Error:** Something is terribly broken or misconfigured. Report this error to the E-Center mailing list.
- **502 Bad Gateway:** E-Center service is down.

The E-Center DRS API follows REST principles as closely as possible. There is no default format for the response encoding. The service will send back **404 Error** code if format is not set in the request. One has to specify in what format the body of response is requested. For example by adding **.json** to the end of the requested resource path one may expect to receive **JSON** encoded response. The client 's API should utilize corresponded **JSON** parsing facility available for the language of choice. Or it may request **XML** or **YAML** formatted data structure by adding **.xml** or **.yaml** correspondently. Please note all examples of the DRS responses in this document are formatted for the case of **JSON** encoding. The **JSON** encoding is used by the E-Center front-end.

The easiest way to get data from DRS is to use **curl** command line tool available on any UNIX platform. Or utilize any HTTP protocol API library for particular language of choice – perl, python, java...etc.

Data Retrieval Protocol

Each request is constructed as a specially formed URI with supplied string of parameters. The most basic resource is **data** time series.

As was noted above, in case of error the corresponded HTTP status will be returned (500 for example) and instead of normal response the JSON encoded error message could be utilized by client application:

Next results are for the example where **src_ip** and **dst_ip** are supplied as and the possible traceroute command would result in this trace:

Traceroute:

from - 192.12.15.26(lhcprefmon.bnl.gov)

to - 131.243.24.11(nettest.bnl.gov)

```
traceroute to 131.243.24.11 (131.243.24.11), 30 hops max, 140 byte packets
 1 mutt.usatlas.bnl.gov (192.12.15.224)  0.538 ms  0.633 ms  0.628 ms
 2 amon.bnl.gov (130.199.3.124)  0.476 ms  0.469 ms  0.593 ms
 3 bnlmr2-bnlsite.es.net (198.124.216.189)  0.424 ms  0.424 ms  0.416 ms
 4 aofacr2-ip-bnlmr2.es.net (134.55.217.142)  2.566 ms  2.548 ms  2.533 ms
 5 bostcr1-ip-aofacr2.es.net (134.55.41.121)  7.460 ms  7.459 ms  7.444 ms
 6 cleocr1-ip-bostcr1.es.net (134.55.41.145)  20.577 ms  20.609 ms  20.524 ms
 7 chiccr1-ip-cleocr1.es.net (134.55.217.54)  29.848 ms  29.827 ms  29.827 ms
 8 kanscr1-ip-chiccr1.es.net (134.55.221.57)  40.235 ms  40.205 ms  40.208 ms
 9 denvcr2-ip-kanscr1.es.net (134.55.209.45)  53.373 ms  53.345 ms  53.350 ms
10 sunncr1-denvcr2.es.net (134.55.220.50)  80.493 ms  80.503 ms  80.479 ms
11 sunnsdn2-sunncr1.es.net (134.55.209.97)  80.458 ms  80.415 ms  80.457 ms
12 slacmr2-ip-sunnsdn2.es.net (134.55.217.1)  80.856 ms  80.831 ms  80.820 ms
13 lblmr2-ip-slacmr2.es.net (134.55.219.9)  81.723 ms  81.674 ms  81.559 ms
14 lbln-ge-lblmr2.es.net (198.129.224.1)  82.135 ms  82.091 ms  82.090 ms
15 ir2gw.lbl.gov (131.243.128.12)  82.245 ms  82.058 ms  82.042 ms
16 nettest.lbl.gov (131.243.24.11)  81.734 ms  81.728 ms  81.722 ms
```

And partial results (truncated for this document to the first 3 network segments in the trace):

```

direct_traceroute: {
  198.124.238.37: {
    1286570606: {
      hop_delay: "0.093",
      metaid: "778",
      hop_num: "1",
      hop_ip: "198.124.238.37",
      ip_noted: "198.124.238.37",
      timestamp: "1286570606"}},
  134.55.41.121: {
    1286570606: {
      hop_delay: "7.089",
      metaid: "778",
      hop_num: "3",
      hop_ip: "134.55.41.121",
      ip_noted: "134.55.41.121",
      timestamp: "1286570606"}},
}
reverse_traceroute: {
  198.129.254.29: {
    1286570736: {
      hop_delay: "0.093",
      metaid: "1324",
      hop_num: "1",
      hop_ip: "198.129.254.29",
      ip_noted: "198.129.254.29",
      timestamp: "1286570736"}},
  134.55.217.2: {
    1286570736: {
      hop_delay: "1.535",
      hop_num: "3",
      metaid_id: "1324",
      hop_ip: "134.55.217.2",
      ip_noted: "134.55.217.2",
      timestamp: "1286570736"}},
},
traceroute_nodes: {
  198.124.238.37: {
    longitude: "-72.8805",
    nodename: "bnl-mr1-bnl-pt1.es.net",
    hub: "bnl-mr2",
    latitude: "40.8703",
    ip_noted: "198.124.238.37",
    netmask: "30"},
  134.55.41.121: {
    nodename: "bostcr1-ip-aofacr2.es.net",
    hub: "bost-cr1",
    latitude: "42.3673",
    ip_noted: "134.55.41.121",
    netmask: "30"}
},
snmp: {
  198.124.238.37 : [
    [ 1286570430, { utilization: 211551400,
                    capacity: 10000000000,
                    errors: 0,
                    drops: 0
                  }
    ],
    [1286570580, { utilization: 199693000
                    capacity: 10000000000,
                    errors: 0,
                    drops: 0
                  }
    ]
  ],
  134.55.41.121 : [
    [1286570430, {utilization: 493038181.33334
                  capacity: 10000000000,
                  errors: 0,
                  drops: 0
                }
    ]
  ]
}

```

```

    },
    [1286570580, {utilization: 2084.47994
                  capacity: 10000000000,
                  errors: 0,
                  drops: 0
                }
    ]
  }
}

```

The main change from previous version is the absence of the *hop_id* and usage of the actual IP of the hop node as the "primary key" among traceroute and SNMP provided data. The *ip_noted* and *hop_ip* entries are duplicates in the hop data and *hop_ip* is left for the historical reason.

Data Request

The request URI construction for data time series retrieval must follow these rules:

```

ecenter_server_hostname := <hostname of the E-Center data service>;
ipv4 := <ipv4 address, dotted>;
datetime := < YYYY-MM-DD%20hh:mm:ss >;
data_type := '/', ['pinger'|'snmp'|'bwctl'|'owamp'|'traceroute'];
response_format := ['.json'|'.xml'|'.yaml'];
data_request := ' GET http://, ecenter_server_hostname , '/data',
data_type?, response_format, '?', ['src_ip=', ipv4|'src_hub=',string], '&', [
'dst_ip=', ipv4|'dst_hub=',string], ('&start=', datetime)?, ('&end=',
datetime)?, ('&resolution=',integer)?, ('&timeout=',integer)?;

```

Where *pinger* for example should be used for requesting data from the PingER service only. The *start* and *end* parameters should be provided to specify time period for the time series. The default value for the *start* is 12 hours ago from the "end" value and for the *end* is now – current time in UTC.

When *data_type* is omitted then DRS will return all available data from all services. The *traceroute* data will be returned for any type of data request.

The *resolution* parameter is optional as well and used to specify number of datapoints to be returned for each type of data requested. The default value is **20** and maximum number is **1000**. The DRS will return average values for the aggregated data points. Another supported optional parameter - *timeout* will help to limit waiting time for results and could be used for a very long time periods, the default value is 120 seconds and maximum value is 1000 seconds.

As one can see there are two distinctive types of the end-point used for the data requests. One is based on the provided IPv4 address and another one is HUB based. Where **HUB** is the identifier used for the aggregated collection of the end-site pS-PS monitoring services along with ESnet pS-PS services located at the border of the end-site. The naming scheme for **HUB** names is following the ESnet Layer2/3 topology scheme.

Data Response

The response for the data request will follow the next format (in case of **JSON** response format, but the data flow is the same for **XML** or **YAML**):

```
data_id := integer;
trace_id := integer;
hop_ip := ipv4;
timestamp := integer;
metric_name :=
['minRtt'|'maxRtt'|'medianRtt'|'minIpd'|'maxIpd'|'meanIpd'|'duplicates'|
'clp'|'iqrIpd'|
'lossPercent'|'throughput'|'jitter'|'lost'|'sent'|'min'|'max'|
'minttl'|'maxttl'|'dups'|'maxerr'|
'max_delay'|'min_delay'|'timestamp'];
snmp_metric := ['capacity'|'utilization'];
hop_metric :=
['ip_noted'|'hop_ip'|'hop_num'|'hop_delay'|'timestamp'|'metaid'];
traceroute_names :=
['ip_noted'|'longitude'|'latitude'|'nodename'|'hub'|'netmask'];
data_result := [integer|float|'NULL'];
hop_result=[data_result|ipv4|timestamp];
metric_stat := [metric_name|snmp_metric], ':', data_result;
hop_stat := hop_names, ':', hop_result;
trace_stat := traceroute_names, ':', hop_result;
data_item := '{' ['direct_traceroute'|reverse_traceroute'], ': { ',
                { hop_ip, ': { ',
                    { timestamp ': {',
                        { hop_stat, { ',', hop_stat }*, '}', ' }*',
                        '}', ' }*', '}', ' ',
                    'traceroute_nodes: {', { hop_ip, ': {',
                        { trace_stat, { ',', trace_stat }*, '}', ' ',
                        { 'snmp: { ', hop_ip, ': [',
                            { timestamp, ': {', metric_stat,
                                { ',', metric_stat }*, '}' }*,
                                ']' },
                            }*,
                        { ['owamp'|'bwctl'|'pinger'], ': {',
                            'src_ip: { ',
                                'dst_ip: { ',
                                    { timestamp, ': {', metric_stat,
                                        { ',', metric_stat }*, '}' }*,
                                    '}'
                                }
                            }, ' }*'
                        },
                    },
                '}';
```

Example of request:

```
GET http://<ecenter server hostname>/data/snmp.json?  
src_ip=131.243.24.11&dst_ip=198.32.44.130&start=2010-06-05  
06:01:02&end=2010-06-05 07:02:01
```

Please note that only utilization (SNMP) data is returned per **hop_ip**. The End-to-End metrics are returned per source or destination pair (IPv4 address or HUB) and for the whole network path.

Service Informational Requests

First service request is the status one. One can check if service is running by sending this:

```
response_format := ['.json'|'.xml'|'.yaml'];  
hub_request := 'GET http://, ecenter_server_hostname , '/status',  
response_format;
```

If everything is fine then it responds with { status : 'ok' } in case of requested JSON encoding.

Or your API will receive a HTTP error code.

There is more complete request/response where several metrics about the DRS health could be requested. It called a *health* request and it represented as:

```
hub_name := string;  
health_request := 'GET http://, ecenter_server_hostname , '/health',  
response_format, ], ('&start=', datetime)?,( '&end=', datetime)?,  
( '&data=', data_type)?,( '&hub=', hub_name)? ;
```

and response is formatted as:

```
domain_name := <domain part of the hostname - to aggregate service  
hosts>  
month := ['01'..'12'];  
year := ['2010'....inf];  
data_table_id := year, month;  
health_response := '{ time_period: {',  
{ data_type, ': {',  
    'start: ',timestamp, ', end: ',timestamp, '}', '*, }, ',  
    'metadata: {', { hub_name, ': {', {  
        data_type, ': {', data_table_id, ': {',  
            'cached_data_count: ',integer,  
            '}',  
            'metadata_count: ',integer,  
            '}', '*,  
        }, '*,  
    }, '*  
' } }';
```

Here is the example of the “*health*” response:

```
{ time_period: { bwctl: { end: "1287069738",
                        start: "1287026538"},
                  traceroute: {end: "1287069738",
                                start: "1287026538"},
                  owamp: {end: "1287069738",
                           start: "1287026538"},
                  pinger: {end: "1287069738",
                            start: "1287026538"},
                  snmp: {end: "1287069738",
                          start: "1287026538"},
                },
  metadata: {es.net: {bwctl: {
                        201103: {
                          cached_data_count: "4",
                        },
                        metadata_count: "780" },
                  traceroute: { 201103: {
                                  cached_data_count: "2",
                                },
                                metadata_count: "879"},
                  owamp: { 201103: {
                              cached_data_count: "406",
                            },
                            metadata_count: "1146" },
                  pinger: { 201103: {
                              cached_data_count: "40",
                            },
                            metadata_count: "1435" },
                  snmp: { 201103: {
                              cached_data_count: "80",
                            },
                            metadata_count: "698" },
                }
  }
}
```

One can notice the **cached_data_count** indicate how many data entries were cached already for some specific domain and month of the year and **metadata_count** is the number of the cached metadata. The later number is important in understanding if E-Center data collection service were able to acquire any metadata at all. If counter is **0** then E-Center will not be able to send any data requests to the remote pS-PS services.

Next is the basic request for the list of the hubs and could be expressed as:

```
hub_request := ' GET http://, ecenter\_server\_hostname , '/hub',
response_format;
```

with response formatted according to:

```
hub_name := <ESnet provided topology id for the HUB - 4 letters short
and uppercase>;
hub_response := '{', { hub_name, ': { hub_name:', hub_name, ',
latitude:', float, ' longitude: ', float, '}', ' }+', '}';
```

Example of the hub request and response:

http://xenmon.fnal.gov:8098/hub.json

```
{ KANS: { longitude: "-94.5822",
        hub_name: "KANS",
        latitude: "39.1008",
        },
  ALBU: { longitude: "-106.647",
        hub_name: "ALBU",
        latitude: "35.0822",
        }
}
```

Next request is used to obtain list of the nodes with available traceroute metadata where node is the originating node for the traceroute – source node:

source_request := ' GET <http://>, [ecenter_server_hostname](#) , '/source', response_format;

and response is formed as:

```
string := < any character string >;
source_response := '[' , { '{ hub:' , string , ' hub_name:', string , '
nodename:', string , ' ip_noted:', ipv4 , ' latitude:', float , ' longitude: ' ,
float , ' netmask:', integer , '},' }+ , ']';
```

To get all available destinations for any of the IPs from the previous response one may use a *destination* request:

destination_ip_request := ' GET <http://>, [ecenter_server_hostname](#) , '/destination/', ipv4, response_format;

and response is formed as:

```
string := < any character string >;
destination_ip_response := '[' , { '{ hub:' , string , ' hub_name:',
string , ' nodename:', string , ' ip_noted:', ipv4 , ' latitude:', float , '
longitude: ' , float , ' netmask:', integer , '},' }+ , ']';
```

Example of the *destination* request and response (the same example could be shown for the *source* request above):

http://xenmon.fnal.gov:8098/destination/134.79.104.208.json

```
[
  {
```

```

longitude: "-72.8805",
hub: "bnl-mr2",
hub_name: "BNL",
nodename: "bnl-owamp.es.net",
ip_noted: "198.124.238.49",
latitude: "40.8703",
netmask: "24"
}
]

```

The **service** request is utilized to get information about all or some pS-PS services cached by the ECenter.

```

condition := '/', ['id'|'ip'|'url'|'name']
service_request := ' GET http://, ecenter\_server\_hostname , '/service',
condition?, response_format;

```

Without any conditions it will return all services. And with any condition supplied it will try to match this condition with particular attribute of the service. In case of successful match it will return the same data structure but for the shorten list of services.

```

service_type := ['pinger'|'traceroute'|'snmp'|'owamp'|'bwctl'|'hLS'];
datetime := < YYYY-MM-DD%20hh:mm:ss >;

```

```

service_data := '{ service:', integer, ', name:', string, ', url:', string, ',
type:', service_type, ', comments:', string, ', created:', datetime, ',
updated:', datetime, ', ip_noted:', ip_v4, ', is_alive:', ['0'|'1'], '}'
service_response := '[' , service_data, { ',', service_data }* ']';

```

Example of the service request and response:

<http://xenmon.fnal.gov:8098/service.json>

```

[
  {
    name: "ESnet Home Lookup Service",
    service: "1",
    comments: "ESnet Home Lookup Service (Berkeley, CA. USA)",
    created: "2010-07-06 15:55:17",
    is_alive: "1",
    ip_noted: "198.129.254.242",
    url: http://ps1.es.net:8095/perfSONAR\_PS/services/hLS,
    updated: "2010-07-06 15:55:36",
    type: "hLS",
  }
]

```

Appendix

E-Center DRS Database scheme

The major difference in the DB scheme from version 2.3 is new sharding based on the timestamp for the measurements data. Each data table is sliced per month and named accordingly.

For each new month at 00:00:01 am on the first day of month the DB creation script will add new data table for each metric (*snmp*, *pinger*, *owamp*, *bwctl*, *traceroute*) and will generate Object Relational Model for the perl data API. Below is the example template for the DB creation. In the template below the "datestamp" is resolved to the YYYYMM format - for example 201102.

```
--
-- topology hub ( something with coordinates, name )
--
--
CREATE TABLE hub (
hub varchar(32) NOT NULL,
hub_name varchar(32) NOT NULL,
description varchar(100) NOT NULL,
longitude float NULL,
latitude float NULL,
PRIMARY KEY (hub)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='topology hub (esnet)';
--
-- topology layer2 info
--
--
CREATE TABLE I2_port (
I2_urn varchar(512) NOT NULL,
description varchar(100) NOT NULL,
capacity bigint unsigned NOT NULL,
hub varchar(32) NOT NULL,
PRIMARY KEY (I2_urn ),
FOREIGN KEY ( hub ) REFERENCES hub ( hub ) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps topology layer2 info';
--
-- topology layer2 linkage
--
--
CREATE TABLE I2_link (
I2_link bigint unsigned AUTO_INCREMENT NOT NULL,
I2_src_urn varchar(512) NOT NULL,
I2_dst_urn varchar(512) NOT NULL,
PRIMARY KEY (I2_link),
FOREIGN KEY ( I2_src_urn ) REFERENCES I2_port ( I2_urn ) on DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY ( I2_dst_urn ) REFERENCES I2_port ( I2_urn ) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps topology layer2 links';
--
-- nodes - all of them
-- ip_addr supports ipv4 and ipv6 addresses in binary form
--
-- it holds topology info as well via layer2 id ( as http://ogf.org/schema/network/topology/base/20070828/ schema
describes)
--
-- it set as INET6_PTON('131.225.1.1') and essentially a 16 byte representation of the IP
-- it allows indexing and netblock search, to get original IP address - INET6_NTOP(ip_addr)
-- to see dotted form of ipv4 or ipv6 - select ip_noted
--
--
CREATE TABLE node (
ip_addr varbinary(16) NOT NULL,
nodename varchar(255) NULL,
ip_noted varchar(40) NOT NULL,
netmask smallint(3) NOT NULL default '24',
PRIMARY KEY (ip_addr),
KEY (nodename),
KEY (ip_noted),
KEY (netmask)
```

```

) ENGINE=InnoDB CHARSET=latin1 COMMENT='nodes';
--
-- topology layer3 mapping to layer2,
-- use netmask from the node table to get all addresses from the block
-- created and updated are here to provide versioning - the same ip_addr maps to l2_urn in some period of time
--
CREATE TABLE l2_l3_map (
l2_l3_map bigint unsigned AUTO_INCREMENT NOT NULL,
ip_addr varbinary(16) NOT NULL,
l2_urn varchar(512) NOT NULL,
created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
updated TIMESTAMP NOT NULL DEFAULT 0,
PRIMARY KEY (l2_l3_map),
UNIQUE KEY ip_l2_time (ip_addr, l2_urn, updated),
FOREIGN KEY (l2_urn) REFERENCES l2_port ( l2_urn ) on DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY ( ip_addr ) REFERENCES node ( ip_addr ) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps topology layer2-layer3 mapping';
--
-- list of keywords, stores most recent regexp used to obtain that keyword as well
--
CREATE TABLE keyword (
keyword varchar(255) NOT NULL,
pattern varchar(255) NULL,
PRIMARY KEY (keyword)
) ENGINE=InnoDB CHARSET=latin1 COMMENT='project keywords';
--
-- operational table for the most recent status of the service
-- all services are here
--
CREATE TABLE service (
service varchar(255) NOT NULL,
name varchar(255) NOT NULL,
ip_addr varbinary(16) NOT NULL,
comments varchar(255) NULL,
is_alive boolean,
created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
updated TIMESTAMP NOT NULL DEFAULT 0,
PRIMARY KEY (service),
KEY is_alive (is_alive),
FOREIGN KEY (ip_addr) REFERENCES node (ip_addr)
) ENGINE=InnoDB CHARSET=latin1 COMMENT='ps-ps services operational table';
--
-- list of eventtypes, metadata will refer to the ref_id here, not the SERVICE !
--
CREATE TABLE eventtype (
ref_id bigint unsigned AUTO_INCREMENT NOT NULL,
eventtype varchar(255) NULL,
service varchar(255) NOT NULL,
service_type varchar(32) NOT NULL DEFAULT 'hls',
PRIMARY KEY (ref_id),
UNIQUE KEY eventtype_service_type (eventtype, service, service_type),
FOREIGN KEY (service) REFERENCES service (service) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB CHARSET=latin1 COMMENT='eventtypes - many per service - many services for the md';
--
-- keywords_service for many - many rel
--
--
CREATE TABLE keywords_service (
ref_id bigint unsigned AUTO_INCREMENT NOT NULL,
keyword varchar(255) NOT NULL,
service varchar(255) NOT NULL,
PRIMARY KEY (ref_id),
UNIQUE KEY key_service (keyword, service),
FOREIGN KEY (keyword) REFERENCES keyword (keyword) on DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (service) REFERENCES service (service) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB CHARSET=latin1 COMMENT='many to many for keywords_services';
--
-- metadata for the service
-- keeping XML in the subject and parameters but extracting end-to-end data
--
-- src - source or interface address depending on type of the service
-- dst - destination address

```

```

-- direction - for the SNMP metadata - direction of the interface
-- created - updated - to signify active metadata
--
--
CREATE TABLE metadata (
  metaid bigint unsigned AUTO_INCREMENT NOT NULL,
  src_ip varbinary(16) NOT NULL,
  dst_ip varbinary(16) NOT NULL DEFAULT '0',
  direction enum('in','out') NOT NULL default 'in',
  eventtype_id bigint unsigned NOT NULL,
  subject varchar(1023) NOT NULL DEFAULT "",
  parameters varchar(1023) NULL,
  created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated TIMESTAMP NOT NULL DEFAULT 0,
  PRIMARY KEY (metaid),
  KEY (metaid),
  UNIQUE KEY md_ips_type (src_ip, dst_ip, eventtype_id),
  FOREIGN KEY (eventtype_id) REFERENCES eventtype(ref_id) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB CHARSET=latin1 COMMENT='ps-ps metaid provided by each service-eventtype';
--
-- NEXT TABLES FOR THE DATA CACHE
-- we are going to store pinger, owamp, bwctl, snmp data
-- based on source -> destination pairs
--
--
-- pinger data storage, base for shard - each month of data will have own table - pinger_data_201011 for example
--
--
CREATE TABLE pinger_data_[% datestamp %] (
  pinger_data bigint unsigned AUTO_INCREMENT NOT NULL,
  metaid BIGINT unsigned NOT NULL,
  minRtt float NOT NULL DEFAULT '0.0',
  meanRtt float NOT NULL DEFAULT '0.0',
  medianRtt float NOT NULL DEFAULT '0.0',
  maxRtt float NOT NULL DEFAULT '0.0',
  timestamp bigint(12) unsigned NOT NULL,
  minIpd float NOT NULL DEFAULT '0.0',
  meanIpd float NOT NULL DEFAULT '0.0',
  maxIpd float NOT NULL DEFAULT '0.0',
  duplicates tinyint(1) NOT NULL DEFAULT '0',
  outOfOrder tinyint(1) NOT NULL DEFAULT '0',
  clp float NOT NULL DEFAULT '0.0',
  iqIpd float NOT NULL DEFAULT '0.0',
  lossPercent float NOT NULL DEFAULT '0.0',
  PRIMARY KEY (pinger_data),
  KEY (timestamp),
  UNIQUE KEY meta_time (metaid, timestamp),
  FOREIGN KEY (metaid) REFERENCES metadata(metaid) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB CHARSET=latin1 COMMENT='ps-ps pinger data cache';
--
-- BWCTL data storage - base for shard - each month of data will have own table - bwctl_data_201011 for example
--
--
CREATE TABLE bwctl_data_[% datestamp %] (
  bwctl_data bigint unsigned AUTO_INCREMENT NOT NULL,
  metaid BIGINT unsigned NOT NULL,
  timestamp bigint(12) unsigned NOT NULL,
  throughput float default NULL,
  PRIMARY KEY (bwctl_data),
  KEY (timestamp),
  UNIQUE KEY meta_time (metaid, timestamp),
  FOREIGN KEY (metaid) REFERENCES metadata(metaid) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps bwctl data cache';
--
--
-- OWAMP data storage - base for shard - each month of data will have own table - owamp_data_201011 for
example
--
--
CREATE TABLE owamp_data_[% datestamp %] (
  owamp_data bigint unsigned AUTO_INCREMENT NOT NULL,
  metaid BIGINT unsigned NOT NULL,
  timestamp bigint(12) unsigned NOT NULL,
  min_delay float NOT NULL DEFAULT '0.0',
  max_delay float NOT NULL DEFAULT '0.0',

```

```

sent int unsigned NOT NULL DEFAULT '0',
loss int unsigned NOT NULL DEFAULT '0',
duplicates int unsigned NOT NULL DEFAULT '0',
PRIMARY KEY (owamp_data),
KEY (timestamp),
UNIQUE KEY meta_time (metaid, timestamp),
FOREIGN KEY (metaid) REFERENCES metadata(metaid) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps owamp data cache';
--
--      SNMP data storage - base for shard - each month of data will have own table - snmp_data_201011 for example
--
CREATE TABLE snmp_data_[% datestamp %] (
  snmp_data bigint unsigned AUTO_INCREMENT NOT NULL,
  metaid BIGINT unsigned NOT NULL,
  timestamp bigint(12) unsigned NOT NULL,
  utilization float NOT NULL DEFAULT '0.0',
  errors int unsigned NOT NULL DEFAULT '0',
  drops int unsigned NOT NULL DEFAULT '0',
  PRIMARY KEY (snmp_data),
  KEY (timestamp),
  UNIQUE KEY meta_time (metaid, timestamp),
  FOREIGN KEY (metaid) REFERENCES metadata(metaid) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps snmp data cache';
--
-- per hop info - base for shard - each month of data will have own table - hop_data_201011 for example
--
CREATE TABLE hop_data_[% datestamp %] (
  hop_id bigint unsigned AUTO_INCREMENT NOT NULL,
  metaid bigint unsigned NOT NULL,
  hop_ip varbinary(16) NOT NULL,
  hop_num tinyint(3) NOT NULL DEFAULT '1',
  hop_delay float NOT NULL DEFAULT '0.0',
  timestamp bigint(12) unsigned NOT NULL,
  PRIMARY KEY (hop_id),
  UNIQUE KEY meta_time (metaid, hop_ip, timestamp),
  FOREIGN KEY (metaid) REFERENCES metadata(metaid) on DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (hop_ip) REFERENCES node(ip_addr) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps traceroute hops';

```